Docket No. SJO920010075US1
Firm No. 0037.0018

METHOD, SYSTEM, AND PROGRAM FOR TESTING A BUS INTERFACE

BACKGROUND OF THE INVENTION

1. Field of the Invention

[0001] The present invention relates to a method, system, and program for testing a bus interface.

2. Description of the Related Art

[0002] The Peripheral Component Interconnect (PCI) bus architecture provides a low latency path through which devices implementing the PCI architecture can communicate. Details of the PCI bus architecture are described in the publication "PCI Local Bus Specification," Revisions 2.2 (Dec. 1998), published by the PCI Special Interest Group, which publication is incorporated herein by reference in its entirety. Each PCI device that communicates on the PCI bus includes a configuration space including information used to address the device on the PCI bus. During initialization, a device designated as the master processor accesses the PCI bus to detect all the PCI devices present on the PCI bus, builds a consistent address map, and then writes the PCI device base addresses to the configuration space registers of each PCI device. The base address registers define the addresses that other PCI devices on the PCI bus use to communicate with the PCI device to which the addresses are assigned. The base register addresses map into the Input/Output (I/O) space of the device as well as the memory space.

[0003] As part of a power-on self test (POST) during initialization, the master processor will test the PCI devices by reading and writing data to the assigned base addresses in the PCI devices to determine whether the read/write operations complete successfully. During the POST initialization, the master processor also tests the memory and registers of the PCI interface used by the master processor. Prior art PCI devices provide a separate bus interface between the master processor and the memory elements of the PCI interface used by the master processor that is separate from the PCI interface. For instance, in the prior art, the master processor may be

embedded in a PCI card including memory and registers and the PCI bus interface would include a separate non-PCI bus interface on the card between the processor and the memory elements on the PCI card. Additionally, the master processor may be implemented in an Application Specific Integrated Circuit (ASIC) that includes the PCI interface and PCI memory and registers. During initialization, the master processor would use the non-PCI bus interface on the PCI card to test the memory and registers of the PCI interface used by the master processor. After the master processor verifies the accessibility of the base addresses assigned to the external PCI devices as well as the internal memory elements on the PCI card used by the master processor for PCI communication, the master processor would continue with initialization.

[0004] The above prior art initialization architecture requires the use of an additional non-PCI bus interface to test the memory registers of the master processor PCI interface. Further, because an internal interface is used to test the memory elements of the master processor PCI interface, the pins and other PCI interface circuitry between the PCI bus and the master processor PCI interface are not tested because the master processor tests the PCI interface memory elements on the internal non-PCI bus interface.

[0005] For these reasons, there is a need in the art for improved techniques for initializing a PCI or other type of bus interface device.

SUMMARY OF THE PREFERRED EMBODIMENTS

[0006] Provided are a method, system, and program for performing initialization operations in a system including a bus, bus interface and at least one bus device communicating on the bus. The bus interface includes memory capable of being accessed over the bus by the at least one bus device. All bus devices capable of communicating on the bus are detected and each detected bus device and bus interface is configured with base addresses that enable transmission of Input/Output (I/O) requests over the bus to the memory in the bus interface and memory in any bus device including memory accessible over the bus. Testing is performed on the base addresses of the memory in each bus device including memory accessible over the

bus by issuing I/O requests to the base addresses of the memory in each bus device. Memory in the bus interface is tested by issuing I/O requests to the base addresses of the memory in the bus interface over the bus.

[0007] In further implementations, the bus interface includes an initiator and target, and the memory in the bus interface is accessible through the target. Testing the memory in the bus interface further comprises transmitting the I/O requests to the initiator. The I/O requests are transmitted to the bus and the target accesses the I/O requests placed on the bus by the initiator and performs the requested I/O requests. The I/O requests testing the memory on the bus interface test circuitry connecting the target and the bus.

[0008] Still further, the I/O requests to the memory of the bus interface comprise internal wrap signals between the initiator and target on the bus interface.

[0009] The bus interface, bus, and bus devices may implement the Peripheral Component Interconnect (PCI) architecture.

[0010] In still further implementations, the bus interface comprises a bridge between a primary bus and secondary bus. In such case, detecting and configuring all the bus devices comprises detecting and configuring all the bus devices capable of communicating on the primary bus and the secondary bus.

[0011] Further provided are a method, system, and program for performing a verification of a bus interface including an embedded device and memory. The bus interface enables communication with a bus. The bus interface memory is capable of being accessed by one bus device communicating over the bus and the embedded device uses the bus interface to communicate on the bus. The bus device tests the memory in the bus interface by issuing Input/Output (I/O) requests to the memory in the bus interface over the bus. The embedded device tests the memory in the bus interface by issuing Input/Output (I/O) requests to the memory in the bus interface over the bus, whereby the tests performed by the bus device and embedded device test whether the bus interface is capable of handling requests from multiple bus devices over the bus.

[0012] Described implementations provide a technique for testing the operability of memory devices within a bus interface using I/O requests transmitted using the lines connecting the bus and bus interface.

BRIEF DESCRIPTION OF THE FIGURES

[0013] Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 illustrates a PCI interface in which a PCI internal wrap signal is generated;

FIG. 2 illustrates a PCI architecture in which aspects of the invention are implemented;

FIG. 3 illustrates logic to initialize the PCI architecture shown in FIG. 2 in accordance with implementations of the invention;

FIG. 4 illustrates an additional PCI architecture in which aspects of the invention are implemented;

FIG. 5 illustrates logic to initialize the PCI architecture shown in FIG. 4 in accordance with implementations of the invention;

FIG. 6 illustrates a PCI verification system in which aspects of the invention are implemented; and

FIG. 7 illustrates logic to test and verify the design of the PCI architecture shown in FIG. 6 in accordance with implementations of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0014] In the following description, reference is made to the accompanying drawings which form a part hereof and which illustrate several embodiments of the present invention. It is understood that other embodiments may be utilized and structural and operational changes may be made without departing from the scope of the present invention.

[0015] Described implementations utilize a wrap signal to test the memory elements on a PCI interface used by a master processor during initialization of a PCI bus system. FIG. 1 illustrates a PCI interface 2 in which a wrap signal is

implemented. The PCI interface includes initiator function 4 hardware to send read and write requests to other PCI devices (not shown) over a PCI bus 6. The target function 8 includes hardware to receive read and write requests asserted on the PCI bus 6 that are addressed toward the address space of the PCI interface 2. The PCI interface 2 includes off-chip drivers (DRV) to transmit signals to the PCI bus 6 and on-chip receivers (RCV) to receive signals from the PCI bus 6. A wrap signal is transmitted from the initiator function 4 to the target function 8 on the same PCI interface 2. To generate the wrap signal, the initiator function 4 issues the FRAME# command to indicate the beginning of a data transfer operation and issues the IRDY# command to indicate that the initiator function 4 is ready to transfer data. Upon decoding the AD lines on the PCI bus 6 and detecting a request to one base address in the target function 8 configuration space, the target function 8 would assert the DEVSEL# command to access the request on the PCI bus 6. The target function 8 would then issue the TRDY# command to indicate that the target 8 is ready to complete the data transaction asserted on the PCI bus 6.

[0016] A wrap signal comprises a data request transmitted from the initiator function 4 to the target function 8 on the same PCI interface 2. The bold lines in FIG. 1 illustrate the signal paths of the wrap signal. The initiator function 4 would transmit the bus command C/BE# to the PCI bus 6 that is addressed to an address in the target function 8 configuration space. In response, the target function 8 would issue the DEVSEL# to take control over that request on the PCI bus 6. In this way, the initiator function 4 issues wrap signals to send read and write commands to the target function on the same PCI interface 2 by addressing the requests to the base addresses in the configuration space of the target function 8. By using the wrap signal, the initiator function 4 is communicating with the target function 8 using the same PCI interface 2 components that another PCI device on the PCI bus 6 would use to communicate with the target function 8.

[0017] FIG. 2 illustrates a PCI implementation in which aspects of the invention are implemented. A PCI interface 50 includes an embedded master processor 52 that is used to initialize and configure the address space of one or more PCI devices on the PCI bus 54, e.g., PCI device 56. The PCI interface further includes a PCI initiator 58

to communicate PCI read and write commands to the PCI bus 54 and a target 60 that receives read and write requests from the PCI bus 54 that target base addresses in the configuration space of the target 60. A Direct Memory Access (DMA) Engine 53 handles Input/Output requests between the master processor 52 and initiator 58. The base addresses in the configuration space of the target define addressable locations in the memory components accessible through the target 60, including an SRAM 62, registers 64, and SDRAM 66. The registers 64 may include the configuration space for the target 60. The master processor 68 may also access an external memory device 68 for operations.

[0018] FIG. 3 illustrates logic implemented in the master processor 52 to configure the PCI bus 54 and PCI interface 50. Control begins at block 80 with the master processor 52 accessing the PCI bus 54 to detect all the PCI devices 56 that communicate on the PCI bus 54. The master processor 52 then builds (at block 82) an address map for all the PCI devices 56 on the PCI bus 54. For each determined PCI device 56, the master processor 52 issues (at block 84) PCI write configuration commands to write the base addresses in the determined address map to the configuration space of each PCI device 56, including the PCI interface 50 used by the master processor 52 to access the PCI bus 54. The master processor 52 then issues (at block 86) PCI write and read commands to write and then read data to the base addresses of memory locations in the PCI devices 56 to test their operation. If (at block 88) the write/read test to any addressable location in the configuration space of any PCI device failed, i.e., the data read differed from the data written, then the master processor 52 generates (at block 90) an error message indicating the PCI device(s) that failed the write/read diagnostic test. This error message may further indicate the base addresses at which the error occurred. From block 90, control may proceed to block 94 to continue testing.

[0019] If (at block 88) all the write/read tests succeeded, then the master processor 52 sends (at block 92) a PCI write command through the initiator 58 to the PCI bus 54 that targets an address in the target 60 configuration space. In response, the target 60 would assert control over the request on the PCI bus 54 and write the requested data to the specified base address location in one target memory location 62, 64, 66.

After writing the data to a memory location accessible to the PCI bus 54 through the target 60, the master processor 52 would issue (at block 94) a PCI read command to the PCI bus 54 to read the data from the address locations to which data was written. If (at block 96) the read data is not the same as the data written to any tested addressable location, then the master processor 52 would generate (at block 98) an error message indicating the target addresses where the error occurred. If (at block 96) the data read from addressable locations in the memory elements 62, 64, 66 behind the target 60 is the same as the data written, then the master processor 52 proceeds (at block 100) with the configuration knowing that the target 60 memory elements are operational.

[0020] With the logic of FIG. 3, the master processor 52 embedded in the PCI interface 50 uses wrap signals to cause the initiator 56 to communicate with the target 60 over the PCI bus 54. This technique avoids any need for an additional non-PCI bus interface between the master processor 52 and memory elements 62, 64, 66 and further tests the circuitry providing the interface between the target 60 and the PCI bus 54.

[0021] FIG. 4 illustrates an additional architecture implementation where a PCI-to-PCI bridge 250 enables communication between a primary PCI bus 252 and a secondary PCI bus 254 to which one or more PCI devices 256 and 258, respectively, are attached. A master processor 260, which performs initialization and configuration operations, communicates with the PCI-to-PCI bridge 250 through a system bus 262 and host/PCI bridge 264. The PCI-to-PCI bridge 250 includes a primary target 266, which includes hardware to receive communications from the primary PCI bus 252, a First-in-First-Out (FIFO) buffer 268 to buffer requests received at the primary target 266 intended for the secondary PCI bus 254, and initiator hardware 270 to issue read/write requests to the secondary PCI bus 254 to the PCI devices 258 on the secondary PCI bus 254 or to PCI devices on a PCI bus attached to a yet further PCI bridge attached to the secondary PCI bus 254 or a bus below the secondary PCI bus 254. The buffer 268 may be used for speed matching and to avoid overruns. The PCI-to-PCI bridge 250 further includes secondary target hardware 272 providing access to memory elements 274, 276, and 278 in the PCI-to-

PCI-bridge 250 via the secondary PCI bus 254. The target 272 would respond to requests asserted on the secondary PCI bus 254 that are directed toward a base address of one of the memory elements 272, 274, and 276. Additional PCI-to-PCI bridges may be attached to or below the secondary PCI bus 254 providing access to still further PCI busses and PCI devices. The master processor 260 is further capable of accessing a memory device 280 through the host/PCI bridge 264. Further details of the architecture of a PCI-to-PCI bridge are described in the publication entitled "PCI to PCI Bridge Architecture Specification," Revision 1.1 (Dec., 18, 1998), published by the PCI Special Interest Group, which publication is incorporated herein by reference in its entirety.

[0022] FIG. 5 illustrates logic implemented in the master processor 260 to perform an initialization of the PCI-to-PCI bridge 250 and PCI devices 256, 258 attached to the PCI busses 252 and 254 or any other busses below the secondary PCI bus 254. After beginning initialization (at block 300), the master processor 260 accesses (at block 302) the system bus 262 to determine all PCI-to-PCI bridges 250 attached directly or indirectly to the system bus 262, all PCI busses 252 and 254 below any determined bridge, and any PCI device 256 and 258 attached to one of the determined PCI busses 252 and 254. The master processor 260 then assigns (at block 306) PCI bus numbers to each located bridge and builds a consistent map of addresses for located bridges 264, 250 and PCI devices 256, 258 in a manner known in the art. For each determined PCI bridge 250, 264 and PCI device 256, 258, the master processor 260 issues (at block 308) PCI write configuration commands to write the base addresses to the configuration space of each PCI bridge 250, 264 and PCI device 256, 258 accessible through such bridge.

[0023] The master processor 260 then issues (at block 310) write and read commands to the base addresses of the memory in each determined PCI device and PCI bridge. If (at block 312) the write/read test did not succeed for every tested PCI bridge 250, 264 and PCI device 256, 258, i.e., the written data did not match the read data, then an error message is generated (at block 314) indicating the PCI bridge(s) and/or PCI device(s) that failed the test. The addresses where the failure occurred may also be listed. If (at block 312) all the write/read tests passed or from block

314,control proceeds to block 318 where the master processor 260 sends PCI read and write commands to the base addresses assigned to the memory elements 274, 276, and 278 accessed through the secondary target 272. Such commands would be passed through the host/PCI bridge 264 to the primary target 266 over the primary PCI bus 252. The primary target 266 would select the requests from the primary PCI bus 252 and buffer the requests in the FIFO buffer 268. The initiator 270 would then access the commands from the FIFO buffer 268 and transmit the requests to the secondary PCI bus 254. The secondary target 272 would assert control over the request directed toward a base address in one of the memory elements 274, 276, and 278 accessed through the secondary target 272.

[0024] If (at block 320) the data read from all of the targeted memory elements 274, 276, and 278 is the same as what was written, then the master processor 260 proceeds (at block 322) with initialization. Otherwise, if (at block 320) there was a failure at one of the targeted addresses of the memory elements 274, 276, and 278, then an error message is generated (at block 324) indicating the targeted addresses where the error occurred.

[0025] As with the initialization described with respect to FIGs. 2 and 3, the memory elements on the PCI-to-PCI bridge used by the master processor 260 are tested without the need for additional bus interfaces and tested in a manner that checks the circuitry between the secondary PCI bus 254 and the secondary target 272.

[0026] In further implementations, the wrap signal may be utilized in the design verification phase when developing a PCI type device. During PCI design testing, the target in the PCI device must be tested to determine whether the target can handle requests from multiple initiators. In the prior art, to perform such testing, the PCI developer must attach the PCI prototype device to a PCI bus and then attach multiple PCI driver models to the same PCI bus, each including initiator and target hardware. The multiple PCI driver models would then be programmed to initiate requests to the target on the PCI prototype device to test the ability of the target to handle requests from multiple initiators. The design and coordination of the operations of multiple PCI drivers, including testing of posted writes and delayed reads, requires considerable development effort.

[0027] FIG. 6 illustrates an implementation of a PCI interface and PCI bus architecture similar to the PCI interface architecture described with respect to FIG. 1, where the PCI interface 350 includes components 352, 358, 360, 362, 364, and 366 similar to the components 52, 58, 60, 62, 64, and 66. In the described implementations, the verification architecture includes a PCI driver model 380 having initiator 382 and target 384 hardware, wherein the initiator 382 is capable of generating read/write requests to test the target 360 over the PCI bus 354. A DMA engine (not shown) may process I/O requests between the master processor 352 and the initiator 358.

[0028] FIG. 7 illustrates logic implemented in the design verification system shown in FIG. 6 to test whether the target 360 can handle I/O requests from multiple initiators. Control begins at block 400 to begin testing the target 360. The embedded master processor 352 would perform (at block 402) the POST and initialization operations to configure the address space for the PCI bus 354 and PCI driver model 380. The master processor 352 and PCI driver model initiator 382 are programmed to issue (at block 404) PCI write commands to write data to the base addresses of the memory elements 362, 364, and 366, which may include posted writes. After the writes, the master processor 352 and PCI driver model initiator 382 issue (at block 406) PCI read commands to the addresses in the memory elements 362, 364, 366 to which the data was written. The read commands from the master processor 352 are sent to the initiator 358. The initiator 358, in turn, sends the read commands directed to the configuration space of the target 360 to the PCI bus 354. The target 360, upon detecting base addresses in the target 360 configuration space on the PCI bus 354, would then assert control over those read commands and read the data from the target addresses in the memory elements 362, 364, and 366 and return the requested data to the PCI bus 354 for return to the embedded processor 352 through the initiator 358. Writes would be processed in a similar manner, in that the writes would be sent to the initiator 358, which in turn sends the writes to the target 360 to apply to the target configuration space. If (at block 408) the data written from the master processor 352 and PCI driver model 380 to the target memory addresses is not the same as that read for all targeted addresses, then the device detecting the error, which may be either the

embedded processor 352 or PCI driver model 380, generates (at block 410) an error message indicating the targeted addresses where the error occurred. If no error occurred, i.e., the written and read data was the same, then the master processor 352 and PCI driver model 380 would generate a signal that no error occurred.

[0029] The described architecture and logic of FIGs. 6 and 7 provides a technique for testing whether a PCI target can handle requests from multiple initiators using an architecture that only requires the use of one PCI driver model 380 having an initiator 382 to issue the test request. With the described implementations, the testing for an additional initiator is performed by the embedded processor 352 using the PCI internal wrap mode. In this way, the described implementations avoid the need for implementing and coordinating multiple PCI driver models.

Additional Implementation Details

[0030] The described logic for performing initialization and test verification operations may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" as used herein refers to code or logic implemented in hardware logic (e.g., an integrated circuit chip, Field Programmable Gate Array (FPGA), Application Specific Integrated Circuit (ASIC), etc.) or a computer readable medium (e.g., magnetic storage medium (e.g., hard disk drives, floppy disks,, tape, etc.), optical storage (CD-ROMs, optical disks, etc.), volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, firmware, programmable logic, etc.). Code in the computer readable medium is accessed and executed by a processor. The code in which preferred embodiments are implemented may further be accessible through a transmission media or from a file server over a network. In such cases, the article of manufacture in which the code is implemented may comprise a transmission media, such as a network transmission line, wireless transmission media, signals propagating through space, radio waves, infrared signals, etc. Of course, those skilled in the art will recognize that many modifications may be made to this configuration without departing from the scope of the present

invention, and that the article of manufacture may comprise any information bearing medium known in the art.

[0031] In the described implementations, the initialization and configuration operations, and internal wrap signal testing, were performed by a processor device. In alternative implementations, any type of device may perform the initialization, such as a Basic Input Operating System (BIOS) component or other hardware device dedicated to initialization or configuration. Alternatively, the processor performing the initialization and configuration may be designed to perform operations unrelated to configuration and initialization. Still further, the initialization operations may be distributed across multiple hardware components and/or processors.

[0032] In the described implementations, an initialization device that initializes the configuration space of the PCI devices uses the internal wrap signal to test the memory on the bus interface or bridge. In alternative implementations, the component that tests the operation of memory within a PCI device using the internal wrap signal may not perform initialization operations. In implementations including a DMA engine between the initiator and master processor, the DMA engine may perform the testing operations, thereby relieving the master processor of such burden.

[0033] In the described implementations, the initialization did not complete if the wrap signal testing of the memory elements failed. In alternative implementations, the initialization may proceed with errors noted.

[0034] In further implementations, the PCI architecture may include additional PCI devices and bridges than those shown in FIGs. 2, 4, and 6.

[0035] The described bus and bridge implementations utilized the PCI architecture. However, in alternative interface implementations, bus and bridge technology known in the art other than PCI may be used to implement the bridge and bus interfaces.

[0036] Certain logic was described as being performed by specific components, such as the processor, initiator, target, PCI driver model, etc. Notwithstanding, described as being implemented within specific components may be implemented elsewhere.

[0037] The described implementations discussed the use of PCI read and write operations to test the operation of addressable memory elements. Any type of read

and write operation may be used to test the memory operation, such as posted writes, delayed reads, delayed writes, etc.

[0038] In the described implementations, the memory in the bridge and bus interface was tested by comparing data read from the memory with what was originally written. In alternative implementations, the memory may be tested in a manner different than comparing data read with what was originally written.

[0039] In the described implementations, the master processor functions as an initialization device that configures the address space of the PCI interface. In alternative implementations, multiple devices or device components may perform the initialization operations described with respect to the master processor.

[0040] The preferred logic of FIGs. 3, 5, and 7 describes specific operations occurring in a particular order. In alternative implementations, certain of the logic operations may be performed in a different order, modified or removed. Morever, steps may be added to the above described logic and still conform to the described implementations. Further, operations described herein may occur sequentially or certain operations may be processed in parallel.

[0041] The foregoing description of the described implementations has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.